Verification of Synchronous Circuits by Symbolic Logic Simulation

Martin Schaef

7th November 2003

1 Verification and Simulation

As a part of formal hardware verification symbolic logic simulation is a usefull method to detect sources of design errors and to proof the correctness of synchronous circuits.

We have to derive strictly *verification* from *simulation*. Simulation means just testing a limited set of test cases and checking if the results correspond to the system specification. Verification is the mathematical proof. In this report simulation is used as a tool for verification.

1.1 Logic Simulators

The correctness of a digital circuit can be shown by a logic simulator, if only circuits implementing the system specification will produce a particular response. Unfortunately, a correct result does not guarantee that our circuit is working, it is also possible that the set of test cases was not large enough.

What we are looking for, is the minimal set of test cases we need, to show that our circuit is working.

1.2 three-valued logic modeling

An important method to shrink our set is the three-valued logic modeling. The set of states {0,1} is augmented by a third value X for an unknown digital value. X is set for any bit, that does not effect evaluation of our circuit. Since it is not important if X is set to 0 or 1, the number of test cases is decreasing for every X we use. So our next goal is, to use as many X as possible, to derive a small set of patterns.

Three-valued logic modeling is supplied by most modern logic simulators.

1.3 Symbolic Simulation

Not all classes of circuits can be verified by just simulation a polynomial number of patterns, e.g addition and parity depend on a large number of

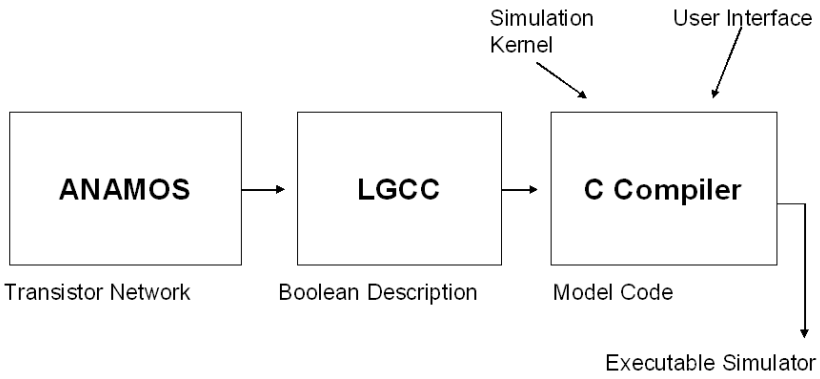

Figure 1: COSMOS program structure

input variables, so the gain of three-valued logic modeling is quite small.

Therefore we use symbolic simulators. The user may introduce a set of Boolean variables to represent input and initial state of the circuit, the simulator computes the behavior of the circuit as a function of these variables.

1.4 Example - COSMOS, a symbolic simulator

A symbolic simulator as shown in figure 1 consists of three parts:

1. *ANAMOS*,

   analyses the switch-level network and returns a Boolean representation of each subnetwork

2. *LGCC*,

   translates the Boolean representation into a set of C language evaluation proceduers.

3. *C Compiler*,

   generates a simulation program for the circuit.

2 Verification Methodology

Since we cannot verify sequential circuits by just testing a certain number of test cases, we use methodology that includes four steps:

1. The desired functionality, expressed as a state machine operating on an abstracted system state.

2. The circuit interface, expressed in terms of the clocking patterns, as well as the times within a clock cycle at which the inputs are applied and the outputs sensed.

3. The mapping between the abstract system state and the internal circuit state.

4. A transistor-level description of the circuit.

Step 1 and 2 define the expected behavior of the stack as a Boolean equation, the desired input/output behavior. These steps are defined without any knowledge of the hardware, also called, the high level specification.

Step 3 compares our first two parts to the internal system state, to find out what our symbolic simulator needs to know, and what information is missing to derive our simulation patterns.

Step 4 uses our symbolic simulator (e.g. COSMOS) to test our simulation patterns.

3 Specification Example

As an example we are using this method to *Verify a nMos Stack*. Our stack is a 1 bit wide and d bits deep push-down stack. The the last element we put on the stack is labled with 1. Now we are using the technic above to verify this stack step by step.

3.1 High Level Specification

Before we can define the behavior of our circuit we have to introduce some predicates:

push(t) : The Stack executes a Push operation on cycle t.

pop(t) : The Stack executes a Pop operation on cycle t.

hold(t) : The Stack executes a Hold operation on cycle t.

Depth(d,t) : There are d ($0 \leq d \leq k$) items on the stack on cycle t.

Stored(i,t,v) : Value $v \in \{0, 1\}$ is stored on location i ($0 \leq i \leq k$) at the end of cycle t

Input(t) : Value $v \in \{0, 1\}$ is supplied to the stack input during cycle t.

Output(t) : Value $v \in \{0, 1\}$ appears on the stack output during cycle t.

Using these predicates, we can define the basic operations, our stack should be able to process. Since this is just an example, we will not take care

Operation	OP1	OP2
Push	1	0
Pop	0	1
Hold	0	0

Figure 2: Control signals

about detail like overflow or poping on an empty stack. Keeping the high level specification as common as possible guarantees that it will hold for any implementation of the stack.

Now we can specify the effect of push, pop and hold operations on the stack:

$Push$:
$Depth(t-1,d) \wedge \forall[1 \leq i \leq d] Stored(i,t-1,v_{i+1}) \wedge Input(t,v_1) \wedge push(t)$
$\Rightarrow$
$Depth(t,d+1) \wedge \forall[1 \leq i \leq d+1] Stored(i,t,v_i)$

Pop :
$Depth(t-1,d) \wedge \forall[1 \leq i \leq d] Stored(i,t-1,v_i) \wedge pop(t)$
$\Rightarrow$
$Depth(t,d-1) \wedge \forall[1 \leq i \leq d] Stored(i,t,v_{i+1}) \wedge Output(t,v_1)$

$Hold$:
$Depth(t-1,d) \wedge \forall[1 \leq i \leq d] Stored(i,t-1,v_i)$
$\Rightarrow$
$Depth(t,d) \wedge \forall[1 \leq i \leq d] Stored(i,t,v_i)$

3.2 Interface Specification

The Stack operations are provided by a pair of two signals, OP1 and OP2. Since we only have one OP input for our stack, the signal is provided in two-phase nonoverlapping clock cycles, divided in Phi1 and Phi2 phase. Figure 2 shows a codeing of the operation.

Important is, that the control signals for clock cycle t are supplied during the Phi2 phase of clock cycle t-1 and the Phi1 phase of clock cycle t, as

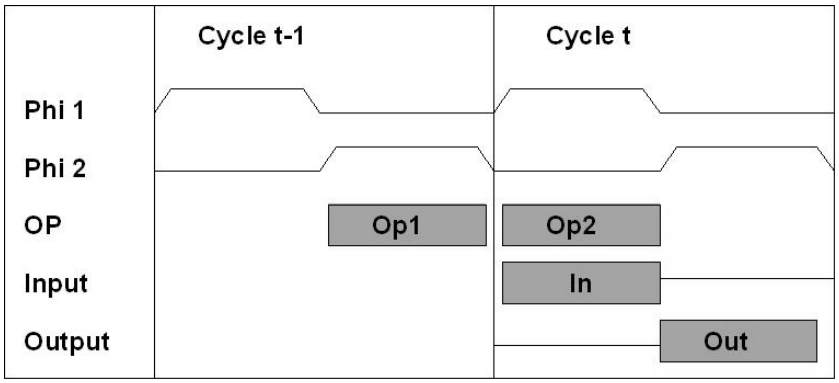

Figure 3: State Circuit Control Interface

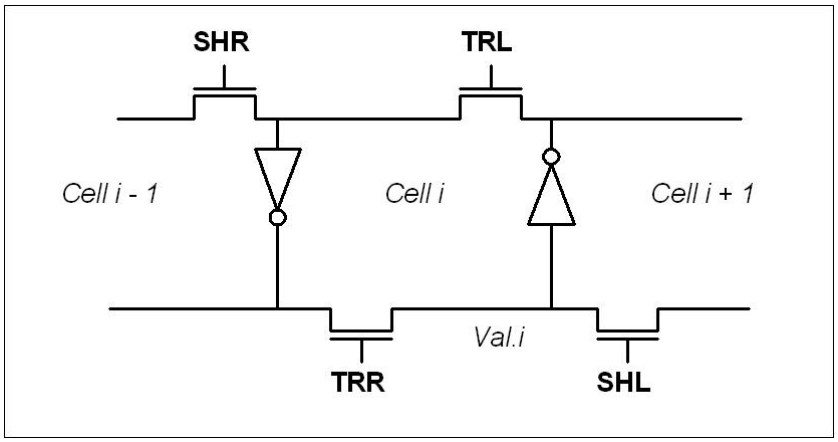

Figure 4: Cell i of Stack

shown in figure 3.

The input to our cell is supplied during the Phi1 phase, and the output during the Phi2 phase.

3.3 State Specification

In figure 4, a stack cell is shown. These cells are orderd from left to right, starting with cell 1.

The transistors controlled by the signals SHR, SHL, TRR, TRL can be considerd as switches, letting the new signal path, if they are set high, and keeping the old one otherwise. On part labled Val.i we can take the negated bit we have stored at the and of a Phi2 phase.

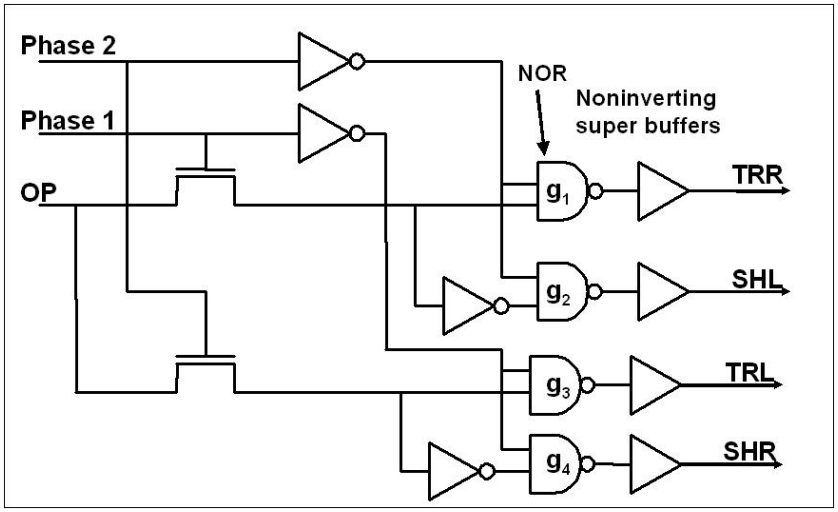

Figure 5: Generating the stack control signals

	Phi 1 Phase				Phi 2 Phase			
	TRR	SHL	TRL	SHR	TRR	SHL	TRL	SHR
Push	0	0	0	1	1	0	0	0
Pop	0	0	1	0	0	1	0	0
Hold	0	0	1	0	1	0	0	0

Figure 6: Control Signals

The control signal for the stack cells are derived by the circuit shown in figure 5. During the Phi1 phase SHR or TRL can be high, the other two are always low. During the Phi2 phase SHL or TRR can be high.

Since our signal in cell i is available after the Phi2 phase, we have to look at two operations at the same time, the operation we are doing and the first part of the following operation. E.g. a Push operation followed by a hold operation would look as follows:

1. *Phi2 Phase of cycle t-1:* TRR is dirven high anything else low
2. *Phi1 Phase of cycle t :* SHR is dirven high anything else low

3. *Phi2 Phase of cycle t :* TRR is dirven high anything else low

We have to remember that the signal representing the content is available at Val.i at the end of the Phi2 Phase, so for our Push operation, our signal is on the left negater (figure 3) at the end of the Phi1 phase of cycle t-1. Setting TRR high leads us to the Val.i part, where we can obtain the result of our last operation. Our signal passes the right negater now and SHR is driven high. Now we have moved one cell to the right. The Phi2 phase of the following operation will lead our signal to Val.i part, where we can read it out. If the next operation is a push or a hold operation, TRR will be driven high and we know what will happen. But what happens if its a pop operation? Then SHL will be driven high, and our signal moves to the Val.i part of cell i-1, after that TRL is high, and our signal moves just to where we need it.

It is important to note, that this implementation does not keep track of the depth of the circuit, so we can introduce a constant k for our stack size. So we can verify our stack for each size k. Since we do not need the predicate depth(d,t) defined in 3.1, we can change our high-level specification to a more common one.

$Push$:
$\forall[1 \leq i \leq k-1] Stored(i, t-1, v_{i+1}) \wedge Input(t, v_1) \wedge push(t)$
$\Rightarrow$
$\forall[1 \leq i \leq k] Stored(i, t, v_i)$

Pop :
$\forall[1 \leq i \leq k] Stored(i, t-1, v_i) \wedge pop(t)$
$\Rightarrow$
$\forall[1 \leq i < k] Stored(i, t, v_{i+1}) \wedge Output(t, v_1)$

$Hold$:
$\forall[1 \leq i \leq k] Stored(i, t-1, v_i)$
$\Rightarrow$
$\forall[1 \leq i \leq k] Stored(i, t, v_i)$

The only information about this specification we still need is the internal representation of the predicate stored.

3.4 Simulation Patterns

Now we can use COSMOS, to derive a symbolic simulator, to test the simulation patterns we can derive by the information from 3.1 to 3.3.

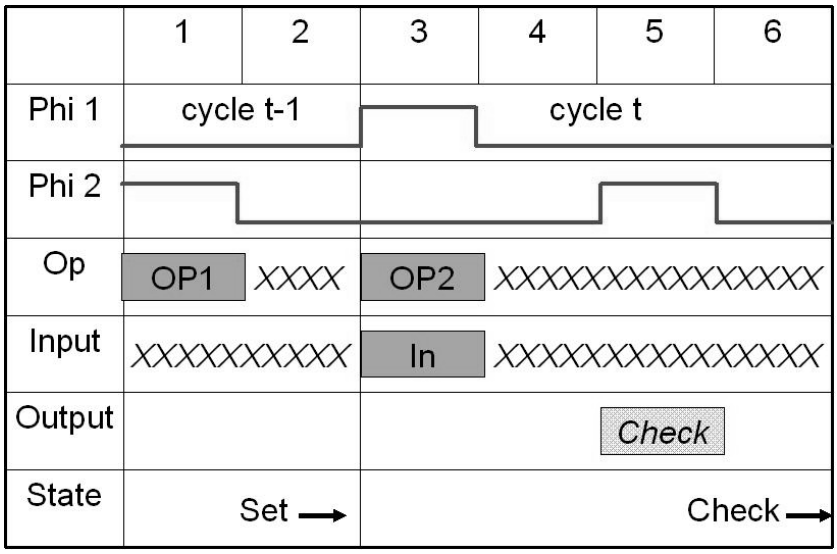

Figure 7: Simulation Patterns

Operation	Settings			Checks		
	In	val.i	val.k	Out	val.i	val.k
	1= i< k			1= i< k		
Push	v_1	$\neg v_{i+1}$	X	-	$\neg v_i$	$\neg v_k$
Pop	X	$\neg v_i$	$\neg v_k$	v_1	$\neg v_{i+1}$	-
Hold	X	$\neg v_i$	$\neg v_k$	-	$\neg v_i$	$\neg v_k$

Figure 8: Stack content

As shown in 3.2 we have to simulate one an a half clock cycle. We are considering 6 steps. Step 1 and 2 for the Phi2 phase of cycle t-1, 3,4 for Phi1 phase of cycle, 5 and 6 for Phi2 phase of cycle t.

Figure 7 shows the simulation patterns for the control circuit. The low levels, e.g. the Phi1 signal during 1 and 2 correspond to logic 0, the high levels, e.g. Phi1 during 3, to logic 1. Double lines indicate a specific variable and X stands for any logic value. *Set* means, that inputs are applied, if we are proceding a push operation. *Check* in step 5 is only used if an output is supplied on a pop operation. *Check* on phase 6 means, that at the end of this phase, our new Val.i is written.

In figure 8 we can see the behavior of the signals stored in the stack cells.

It is important, to remember, that each signal passes 2 negaters during 1 clock cycle.
Now we have all patterns, that need to be simulated.

4 Observation

With this method, we can minimize our set of test patterns. This is realy important for bigger circuits. Perhaps this it not the best way to verify a circuit, but it is very good in detecting errors, making debugging much easier.

Bibliography

Verification of Synchronous Circuits by Symbolic Logic Simulation *Randal E. Bryant - Carnegie Mellon University*

COSMOS - A Compiled Simulator for MOS Circuits *1987 24th ACM/IEEE Design Automation Conference*

Introduction to VLSI Systems *1980 C.A. Mead, L.Conway*